НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 “ Інженерія програмного забезпечення“

на тему Моніторинг прогнозних оцінок ефективності роботи сервісів в складних розподілених системах

Виконав : студент 4 курсу, групи ТІ-61

_____Защик Сергій Миколайович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ___к. т. н., доцент, Гагарін О. О._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 121 “ Інженерія програмного забезпечення “

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

_____Защику Сергію Миколайовичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи Моніторинг прогнозних оцінок ефективності роботи сервісів в складних розподілених систем
керівник роботи Гагарін Олександр Олександрович, к. т. н., доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201__р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи Модуль збору та обробки статистики на мові PHP у середовищі UNIX.
4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) Розробити модуль збору даних, спроектувати формат збереження даних, реалізувати модуль прогнозування на основі збережених даних.
5. Перелік ілюстративного матеріалу Титульний лист, постановка задачі, використані технології, висновки

6. Дата видачі завдання ”11”__жовтня___2019р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Програмна реалізація системи		
5.	Оформлення пояснювальної записки		
6.	Захист програмного продукту		
7.	Передзахист		
8.	Захист		

Студент _____ _____Защик С. М._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Гагарін О. О._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою виконання даної роботи є створення програмного модуля для збору та збереження статистичних даних та прогнозування роботи сервісів у складних розподілених системах.

У результаті виконання роботи було створено систему, що зберігає інформацію про події на сервісах проекту до бази даних, генерує на їх основі прогнозні події та дозволяє візуально спів ставити реальні результати роботи сервіів проекту з прознозними.

Серверна частина реалізована мовами програмування PHP ta Python. Для візуалізації метрик було використано застосунок Grafana. Прогнозування здійснюється за допомогою статистичної моделі ARIMA. База даих – ElasticSearch.

Представлена дипломна робота займає 61 сторінку та включає 26 рисунків, 7 посиланнь і 3 додатки.

Ключові слова: Статистика, метрики, збір метрик, прогнозування, модель ARIMA, шина повідомлень, черга, сервіс, мікросервіс, модуль, сервер, база даних.

ABSTRACT

The purpose of this work is to create a software module for collecting and storing statistics and forecasting the services in complex distributed systems.

The result of the work is creation of a system that stores information about events on services of the project to the database, generates forecast events based on them and allows you to visually compare the real results of work of the project services with the forecast.

The server part is implemented in PHP and Python programming languages. Grafana application was used to visualize the metrics. Forecasting is performed using the statistical model ARIMA. The database is ElasticSearch.

Total capacity: 61 pages, 26 figures, 7 references and 3 appendices.

Tags: Statistics, metrics, metrics collection, forecasting, ARIMA model, message bus, queue, service, microservice, module, server, database.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ARIMA – AutoRegressive Integrated Moving Average

ARMA – AutoRegressive Moving Average

DB – database

QL – query language

СКБД – Система керування базами даних

API – Application Programming Interface

SSL – Secure Sockets Layer

URL – Uniform Resource Locator

ВСТУП

Завдання кожної програмної системи – задовольнити потреби користувача, однак на етапі проектування не завжди можна гарантувати ефективність того чи іншого рішення. Значна частина особливостей системи і варіантів її удосконалення виявляється уже в процесі експлуатації. Саме тому жоден програмний продукт у наш час не обходиться без збору статистики. Інформація про те, як користувачі використовують програму, якому функціоналу надають перевагу а який майже не застосовують, скільки часу витрачають на користування системою – є ключовою для визначення напрямів подальшого розвитку проекту. Визначення цілевої аудиторії та її вимог допомагає пріоритезувати задачі, витрачаючи час розробників найбільш ефективно з точки зору бізнесу. У проектах, що активно розвиваються, не менш важливим є моніторинг впливу кожної зміни, що вносяться у систему, на стабільність роботи продукту. Особливо критичним це питання є у складних багатосервісних системах, що складаються з десятків підсистем. Зв'язані один з одним компоненти формують розгалужену екосистему, у якій найменша зміна однієї складової може непередбачуваним чином вплинути на поведінку іншої.

Складні розроділен системи можуть генерувати тисячі подій щосекундно, тому система збереження даних статистики має високі вимоги до стабільності та швидкодії при пошуку та фільтрації при значних об'ємах зібраних даних. При перебоях в роботі сервісу доступ до цих даних дозволяє пришвидшити пошук джерела проблеми.

Метою даної роботи є створення системи для збору статистики функціонування окремих модулів складного багатокомпонентного продукту та прогнозування їх роботи у майбутньому на основі зібраних даних.

Задачі, що повинні вирішуватися системою, мають бути поділені між окремими структурними модулями: збору даних, прогнозування та візуалізації.

При розробці системи було використано наступні технічні та програмні засоби: Docker-оточення, Nginx-php web-сервер, СКБД ElasticSearch, Docker-контейнер з середовищем Python, шина повідомлень Rabbitmq та застосунок Grafana.

Для опису архітектуру системи та структур збереження даних використано нотацію UML, а саме діаграми прецедентів та класів.

1. ЗАДАЧА РОЗРОБКИ МОДУЛЯ ЗБОРУ СТАТИСТИКИ І ПРОГНОЗУВАННЯ РОБОТИ СЕРВІСІВ

Модуль збору статистики та прогнозування, що розробляється, повинен використовувати спільний інтерфейс збору даних для різних за структурою сервісів та надавати зручне графічне представлення прогнозів на основі зібраної статистики. Система збереження статистичних даних повинна забезпечувати достатню швидкодію при роботі з великими об’ємами даних.

Потенційними користувачами є компанії-розробники складних продуктів-екосистем з сервісною чи мікросервісною архітектурою [1].

1.1 Задача збору статистичних даних

Вибір інтерфейсу збору даних залежить насамперед від архітектури розподіленої системи, специфіки структури сервісів та технологій, що використовуються у продукті. Модуль збору статистики повинен використовувати спільний з іншими сервісами системи спосіб обміну даними. Взаємодія з цим модулем не повинна впливати на продуктивність роботи інших сервісів. Зібрані дані потріюно зберігати в системі на термін, достатній для формування прогнозів на їх основі. Сховище даних має забезпечувати швидкий доступ до великих об’ємів даних з фільтрацією за типами подій, ідентифікаторами сервісів та часом збереженої події. Статистика потрібно вести для конкретних типів подіях на сервісах. Прикладами подій можуть бути успішне проведення оплати у сервісі платежів, успішна реєстрація нового користувача, помилка хмарного сховища при обробці файлу хибного формату або успішно згенерований документ у сервісі онлайн-редактора.

Інформація про кожну подію повинна включати в себе ідентифікатор або код сервісу, тип події, її дату та час, результат роботи (успіх чи помилка) та набір даних, з якими працював сервіс.

1.2 Задача побудови прогнозів

Система повинна містити модель, що на основі даних про подію в минулому прогнозуватиме частоту її появи та результат на день вперед. Модель повинна враховувати сезонність – сталі коливання в часовому ряду, що повторюються через однакові проміжки часу [2]. Навантаження на окремі сервіси продукту зазвичай відрізняється у різний час доби, залежить від дня тижня, початку чи кінця місяця або пори року. Наприклад у будні дні сервіс може використовуватися частіше, у вихідні – рідше, але щотижня коливання повторюватимуться (рисунок 1.1).

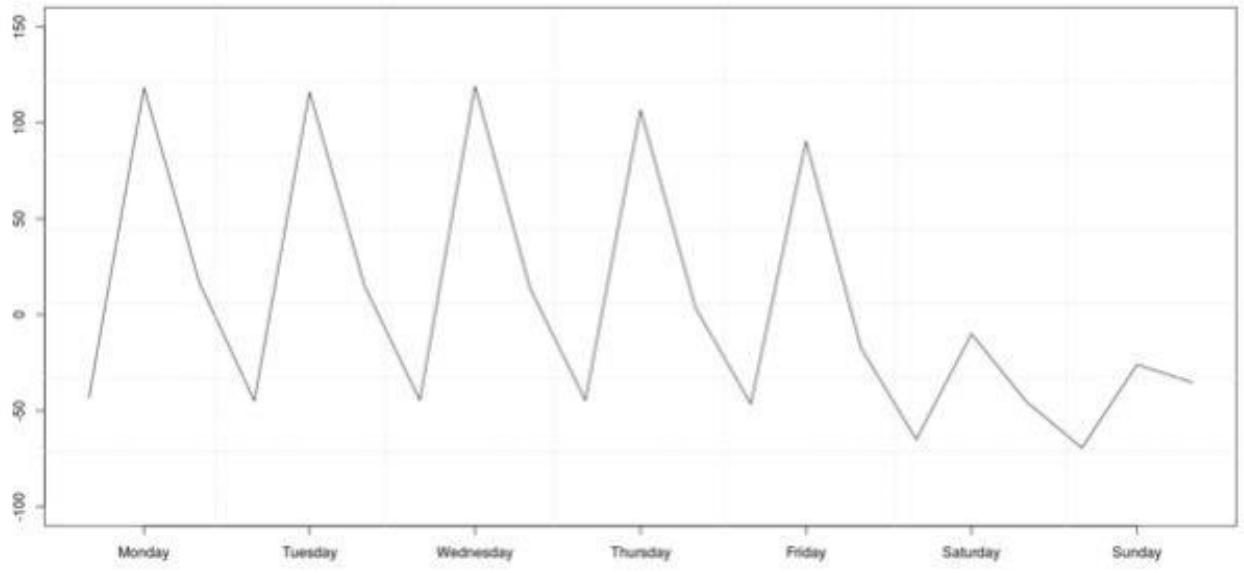

Рисунок 1.1 — Коливання навантаження на продукт протягом тижня

У більшість проектів, що розвиваються, активно залучають нових користувачів, покращують старий функціонал та додають новий. Навантаження та такі системи ростиме з часом, тому воно не є стаціонарним або сезонним. У ньому

спостерігатиметься позитивний тренд — тривалу зміна рівня середнього випадкового процесу [3]. Модель повинна враховувати загальний тренд подій при побудові прогнозу (рисунок 1.2).

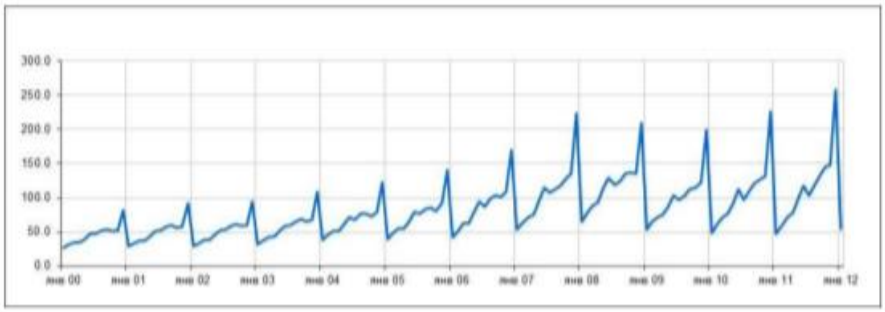

Рисунок 1.2 — Навантаження на сервіси продукту, що розвивається

Тренд використання сервісу може бути як позитивним, так і негативним (рисунок 1.3).

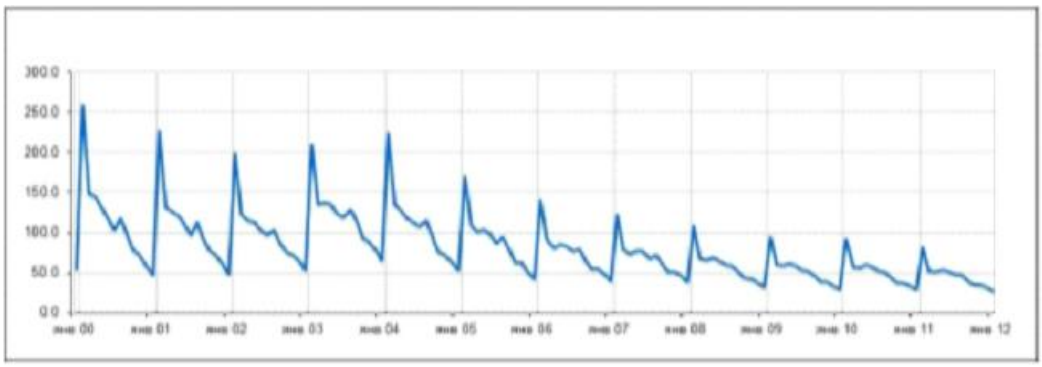

Рисунок 1.3 — Навантаження на сервіси продукту, що популярність якого падає

Причиною цього може бути як загальне падіння попиту на продукт, так і зменшення потреби у конкретному сервісі. Даний часовий ряд [4] містить і сезонні коливання, і негативний тренд.

Модуль прогнозування повинен генерувати продовження часового ряду подій на сервісі на основі попередніх значень ряду, зберігати згенеровані події до сервісу сховища даних.

1.3 Задача візуалізації даних статистики та прогнозів

Реалізації цієї задачі залежить від реалізації збору статистичних даних та від способу формування прогнозу. Дані та прогнози потрібно представити візуально (рисунок 1.4).

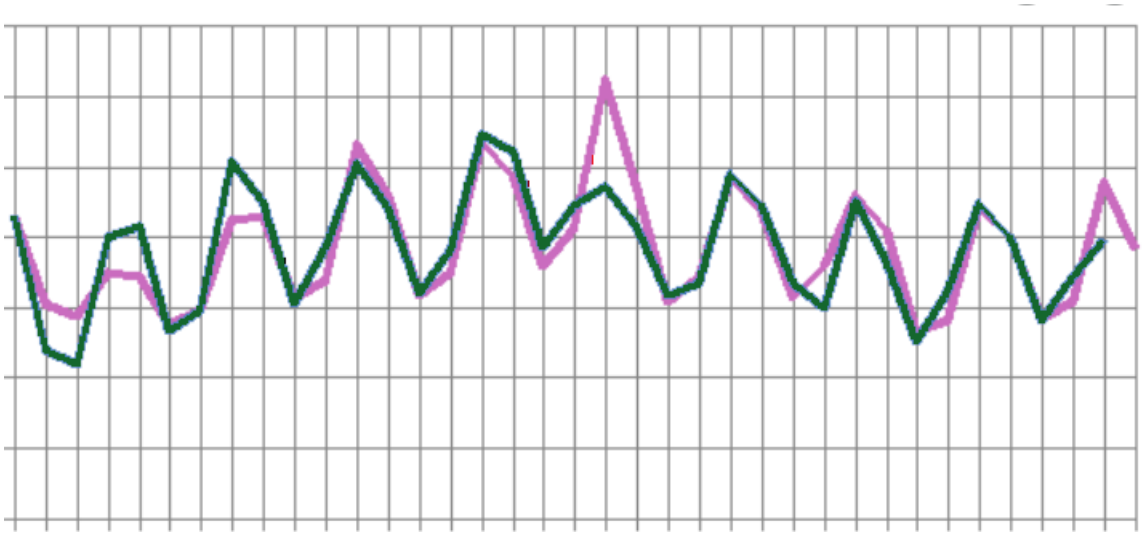

Рисунок 1.4 — Приклад співставлення графіку прогнозу з реальними даними

Окремі графіки повинні бути згрупованими за сервісом, якому вони належать та типом події так, щоб графік прогнозу, зробленого відносно поточного дня, можна було співставити з графіком реальних даних.

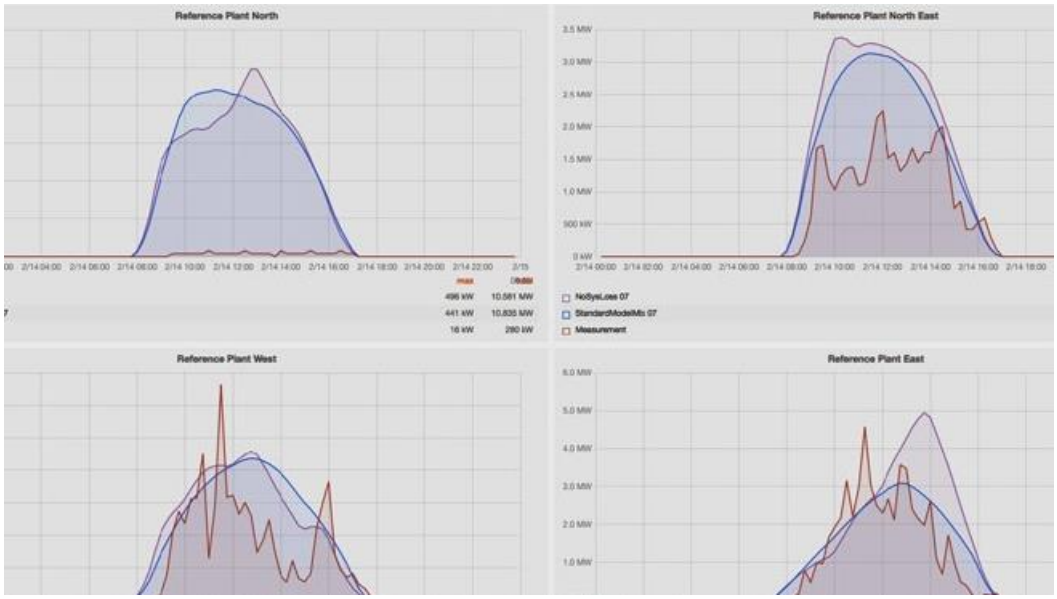

Рисунок 1.5 — Приклад групування графіків за сервісам

Модуль візуалізації даних повинен надавати можливість переглянути детальнішу інформацію про кожну подію часового ряду, на основі якого було побудовано графік та налаштувати на їх основі додаткові фільтри, наприклад події конкретного користувача або платіжні транзакції на суму, що перевищує вказану.

2. ЗБІР СТАТИСТИКИ ТА ПРОГНОЗУВАННЯ В СКЛАДНИХ РОЗПОДІЛЕНИХ СИСТЕМАХ

Проблема збору статистики постає перед кожним проектом, що розвивається. Наявність детальних даних про роботу продукту дозволяє визначати його найбільш використовуваний функціонал, визначати вразливі місця, знаходити ефективну стратегію розвитку продукту, та вчасно реагувати на проблеми у роботі сервісу. Проблема є фундаментальною, тому існує велика кількість як комерційних, так і безкоштовних рішень, однак жодне з них не є універсальним.

Застосування моделей прогнозування дозволяє підвищити ефективність виявлення недоліків у роботі системи. Моніторинг і сповіщення налаштовані реагувати на значні зростання кількості помилок або затримки в роботі сервісів, але коли неполадка не критична різкого скачка помилок може не відбутися, але різниця між прогнозною оцінкою, якої раніше не спостерігалося, вказуватиме на необхідність перевірки даних та пошуку причин розбіжності.

2.1 Збір даних статистики

Збір метрик про роботу сервісу напряму залежить від особливостей його реалізації, технологій, що в ньому використовуються та форматів даних, з якими працює сервіс. Складні розподілені системи можуть включати в себе десятки окремих сервісів, кожен з яких використовуватиме різні набори технологій. Більшість існуючих рішень спеціалізуються на певному функціоналі, наприклад:

- Prometheus – збір метрик
- Datadog – моніторинг і маштабування ресурсів

- Open Web Analytics – збір даних через Web-інтерфейс (HTTP API)
- Kibana – фільтрація та відображення логів

Спеціалізація цих рішень дозволяє застосовувати їх у різних проектах для розв'зання конкретної задачі, однак це створює потребу у використанні комбінації кількох з цих сервісів для повноцінного збору статистики та моніторингу роботи проекту. Альтернативою є створення власного модуля, що буде надавати весь функціонал збору статистики в межах програмної системи, представленої на підприємстві. В такому разі вона буде враховану специфіку набору технологій, деталей реалізації, програмної архітектури та вимог сервісів проекту. Такий модуль забезпечить весь процес обробки метрик та збору даних, збереження їх у сховищі, візуалізацію збереженої інформації. Також це дозволить спростити реалізацію додаткового функціоналу, як прогнозування роботи сервісів продукту.

Система збору статистики, реалізована у вигляді внутрішнього сервісу продукту не потребуватиме HTTP API для збору даних, оскільки матиме доступ до спільної для всіх сервісів проекту шини повідомлень EventBus. Це дозволить уникнути сповільнення сервісів через додаткові запити через API, та мати доступ до даних події напряму. Окрім цього, самі сервіси продукту не потрібно буде модифікувати для додоавання функціоналу надсилання метрик. Збереження статистики в нереляційній СКБД ElasticSearch дозволить візуалізувати їх з допомогою уже існуючих рішень. Модуль прогнозування створений для конкретного продукту враховуватиме його особливості, що підвищить точність прогнозування моделі.

Перевагами створення власного модуля збору та обробки статистики є орієнтованість на конкретний проект з урахуванням його особливостей та архітектури, забезпечення усього необхідного функціоналу обробки та збереження даних і спільна з іншими сервісами проекту інфраструктура.

Недоліком, порівняно з іншими сервісами, є менша універсальність. При інтеграції такого модуля у інший продукт може знадобитись його модифікація, зміна налаштувань або внесення змін в архітектуру системи.

2.2 Прогнозування роботи сервісів

Результатом збору даних статистики роботи сервісів є набір інформації про подій на сервісах з прив'язкою до часу події – часовий ряд подій.

Часові ряди (Time series) дають можливість прогнозувати майбутні значення ряду на основі попередніх значень. Використавши модель прогнозування можна продовжити часовий ряд подій на сервісах. Часові ряди бувають стаціонарними та нестаціонарними. Якщо статистичні властивості певного часового ряду не залежать від часу спостереження, то ряд називається стаціонарним, Тобто статистичні властивості, такі як середнє значення, дисперсія та коваріація (міра лінійної залежності випадкових величин) є не змінюються з часом.

Навантаженість сервісів може залежати від часу доби, дня тижня (вихідний чи будній) або пори року. Крім цього, на неї впливає загальна тенденція розвитку продукту, те, як змінюється кількість користувачів системи з часом. Сезонність у часових рядах – повторювана модель змін за спостережуваний період часу. Наприклад, сервіси обробки документів більш навантажені в будні дні тижня, а піки використання припадають на середину робочого дня. Таким чином кількість подій на сервісі виявлятиме значну сезонність. Часові ряди з трендом (загальною тенденцією росту чи падіння) або із сезонністю є нестаціонарними. Для побудови прогнозу такий ряд потрібно зробити стаціонарним.

Модель ARIMA - класс статистичних моделей для аналізу і прогнозування даних часових рядів. Він обслуговує набір стандартних структур даних часових рядів і надає простий, але потужний метод для створення їх прогнозів [7].

ARIMA - це абревіатура від AutoRegressive Integrated Moving Average. Це узагальнення більш простий метод авторегресивного рухомого середнього (ARMA) і додає поняття інтеграції.

Авторегресивна модель використовує зв'язок між спостереженням і деякою кількістю попередніх спостережень, наприклад між вчорашнім днем та цим днем сім тому.

Інтегрованість моделі означає, що використовується різниця (диференціал) щоб зробити часовий ряд стаціонарним. Якщо диференціювання першого порядку усуває нестаціонарність, то часовий ряд змінюється зі сталою швидкістю. У випадку, якщо потрібно застосувати диференціювання вищих порядків – швидкість зміни ряду не є константною.

Рухома середня (moving average) – для прогнозу використовується середнє значення за попередній проміжок часу.

Кожен з цих компонентів вказано в моделі як параметр. Використовується стандартне позначення ARIMA:

$ARIMA(p,d,q)$,

де p, d, q – цілі числа, що вказують на конкретну використовувану модель ARIMA,

p - число попередніх спостережень, включених в модель (порядок відставання),

d - кількість раз, коли вихідні спостереження диференціюються для отримання стаціонарного часового ряду (порядок диференціювання),

q - Розмір вікна рухомого середнього (порядок рухомого середнього).

Таким чином в основі під модуля прогнозування часового ряду використано модель ARIMA. Він геренерує продовження часового ряду, яке зберігається до бази даних.

2.3 Висновки до розділу

Отже у розроблюваного модуля є вузькоспеціалізовані сервіси-аналоги, що окремо виконують функції збору та обробки метрик, їх зберігання і візуалізції. Для отримання аналогічного функціоналу потрібно буде налаштовувати взаємодію

різних застосунків як з іншими сервісами, так і між собою. Наявність модуля прогнозування, статистичну модель якого оптимізовано для конкретного продукту є перевагою розроблюваної системи. Окрім цього, збір даних статистики відбуватиметься через спільну шину повідомлень, що дозволить не навантажувати сервіси додатквою логікою надсилання даних.

3. ЗАСОБИ РОЗРОБКИ

Ядро під модуля збору статистики написано на мові програмування PHP. Вона надає можливість зручно працювати як з сервісом шини повідомлень Rabbitmq, так і з нереляційною базою даних Elasticsearch. Інфрструктурно модуль розгортається за допомогою Docker. Для візуалізації даних статистики та прогнозів використано сервіс Graphana. Для прогнозування застосовано реалізацію моделі ARIMA мовою програмування Python.

3.1 Подійно-орієнтований підхід

Подійно-орієнтований підхід до розробки програмного забезпечення (з англ. – *event-driven architecture*) – архітектура програмного забезпечення, що базується на управлінні подіями – їх створенні, визначенні, споживанні и реакції на них [5].

Подія – це суттєва зміна стану системи чи одно з її компонентів. Цей архітектурний шаблон може застосовуватися при розробці та реалізації програм і систем, що передають події між слабо пов’язаними компонентами модулями та службами. Система, що створена за такою архітектурою зазвичай містить джерела подій (або агенти) і споживачі подій (або стоки) (рисунок 3.1). Стоки відповідають за реакцію на подію, відповідно отримують і обробляють дані, передані подією. Реакція може повністю або не повністю створюватися стоком. Як приклад, стік може бути відповідальним лише частину обробки події (фільтрація, перетворення подіїі створення нової), делегація події до іншого стоку. Стоки першого типу можуть взаємодіяти зі стандартними методами роботи з даними, а другого – очікують створюють подію і очікують її обробку іншим стоком.

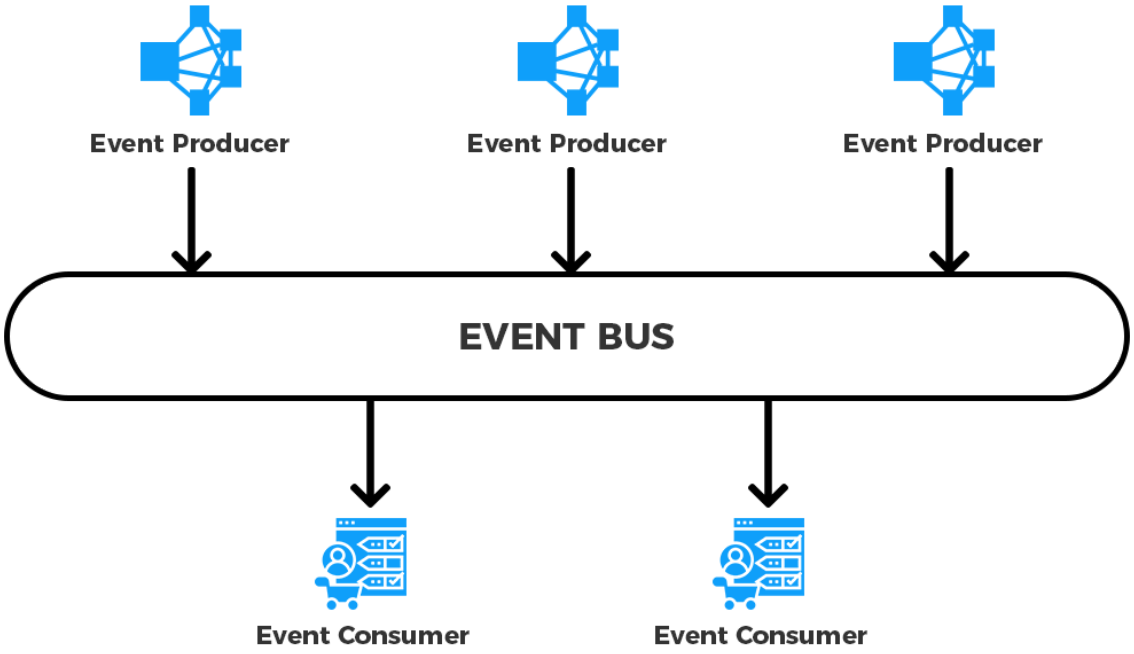

Рисунок 3.1 — схема взаємодії споживачів та генераторів подій

Системи, керовані подіями, є більш орієнтованими на багато поточну та асинхронну роботу з даними.

Сервісний та мікросервісний підходи до проектування програмних продуктів, що зараз заміняють монолітний підхід , створені на основі подійно орієнтованої архітектури.

3.2 PHP, Nginx

PHP (Hypertext Preprocessor (Препроцесор гіпертексту)) – поширена скрипкова мова програмування загального призначення з відкритим вихідним кодом. У сфері веб-розробок PHP є однією з найбільш поширених мов програмування, разом із Java,

.NET, Perl, Python, та Ruby. PHP підтримується переважною більшістю хостинг-провайдерів. Легко інтегрується в Docker інфраструктуру.

PHP включає в себе вбудовані бібліотеки для роботи з MySQL, PostgreSQL, MSSQL, ElasticSearch та ін. Мова забезпечує хорошу швидкодію як web так і для серверних скриптів.

Як HTTP-сервер використовується Nginx. Його основні переваги:

- обслуговування статичних запитів, індексних файлів, автоматичне створення списку файлів
- акселероване проксіювання з підтримкою кешування
- паралельне виконання вкладених запитів на одній сторінці
- підтримка стандарту SSL

3.3 ElasticSearch

Elasticsearch — NoSQL пошуковий сервер бази даних. NoSQL – база даних наступного покоління, що характеризується відмовою від реляційності, розподіленістю, відкритістью сирцевого коду та горизонтальною масштабованістю. Його можна використовувати для пошуку по всіх видів документів. Elastic є розподіленим повнотекстовий пошуковий сервіс з HTTP веб-інтерфейсом і підтримкою безсхемних JSON документів. Застосунок є одним з найпопулярніших пошукових рушіїв. Його розроблено мовою програмування Java, що робить його доступним на платформах, що підтримують JVM.

ElasticSearch забезпечить надійне збереження даних статистики, індексацію та ефективний пошук. Це середовище БД не вимагає попередньо визначеної структури записів, тому підійде для зберігання даних статистики будь-якого з сервісів продукту.

3.4 Docker

Docker є платформою для швидкої розробки, тестування та розгортання застосунків, на основі зліпка (image) процесу у пам’яті. Він упаковує ПО в стандартизовані блоки, які називаються контейнерами. Docker-контейнери містять всі ресурси, потрібні для роботи програми: бібліотеки, системні інструменти, код та середовище виконання. Docker дозволяє швидко масштабувати та розгоотати додатки в будь-якому середовищі [6].

Використання Docker на AWS дає розробникам і системним адміністраторам надійний і економний спосіб компонування, доставки та запуску складних розподілених додатків, незалежно від їх маштабу.

Docker базується на стандартизований підході до виконання коду. Він є операційною системою для контейнерів. Аналогічно до створення віртуальною машиною образу апаратного забезпечення сервера, контейнери створюють віртуальний образ серверної частини операційної системи (рисунок 3.2).

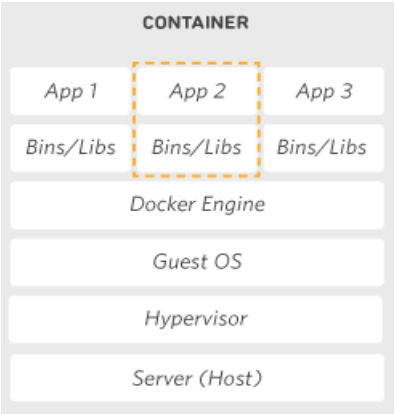

Рисунок 3.2 — Структура DockerEngine

Це дає змогу розгорнути різні додатки без складного налаштування спільної інфраструктури, використовувати хмарні сервіси для запуску контейнерів та спрощує перенесення системи між host-машинами. Окрім цього, контейнери Docker легко маштабуються – Docker може запустити копію контейнера процесу.

3.5 Python

Python - це високорівнева мова програмування загального призначення, яка використовується в тому числі і для розробки веб-додатків. Мова спроектована з ціллю підвищення продуктивності розробника, полегшення розуміння і читання коду.

У Python підтримує присутні кілька парадигм програмування: структурну, об'єктно-орієнтовану, функціональну, імперативну і аспектно-орієнтовану. Мова підтримує динамічну типізацію, автоматичне (garbage collector) керування пам'яттю, механізм обробки виключень (exceptions), багатопоточні обчислення та високорівневі структури даних. Програмний код на Python організовується у функції та класи, які можуть об'єднуватися в модулі, а вони в свою чергу можуть бути об'єднані в пакети. Python зазвичай інтерпретується, але може бути скомпільовано в байт-код Java і в MSIL (в рамках платформ .NET).

За продуктивністю Python схожий на всі інші подібні мови, але можливість компіляції в байт-код дозволяє домогтися більшої продуктивності. На відміну від Ruby і деяких інших мов, Python не дає можливості для змін вбудованих класів, як, наприклад, int, str, float, list і інші.

В Python присутня можливість глобального блокування інтерпретатора (GIL) - при своїй роботі основний інтерпретатор постійно використовує велику кількість потоко-небезпечних даних. Прикладом можуть слугувати словники зі збереженням атрибутів об'єктів, що звертаються до зовнішнього коду. Тому, щоб уникнути колізій та пошкодження даних при взаємодії з ними і спільній модифікації з різних потоків, перед початком виконання таких операцій потік інтерпретатора захоплює GIL, а після закінчення звільняє.

3.6 Rabbitmq

RabbitMQ - це додаток для роботи з чергами повідомлень (шина повідомлень). Програма дозволяє визначати черги до яких можуть підключатися інші додатки, передавати та отримувати повідомлення з черги.

Повідомлення може містити будь-які дані. Наприклад, інформацію про те, який процес розпочати або яку подію створити. Менеджер черг – додаток, який зберігає повідомлення в черзі поки інший додаток (сервер), якому адресовано повідомлення, що не підключитися і не отримає повідомлення з черги і не опрацює його (рисунок 3.3).

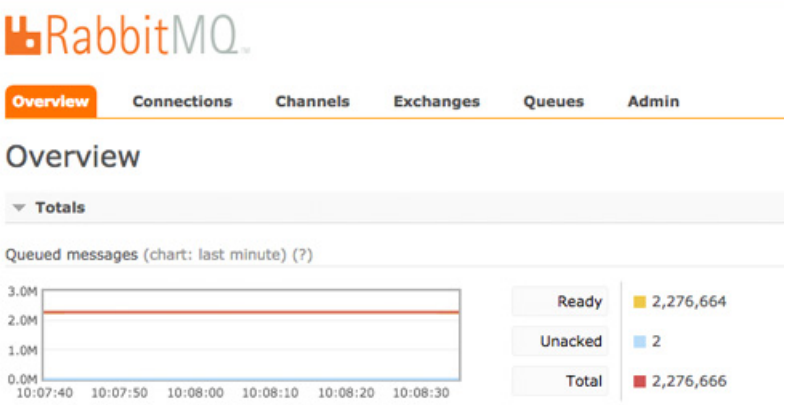

Рисунок 3.3 — Інтерфейс шини повідомлень Rabbitmq

Rabbimq може виступати як прошарок між декількома сервісами, використовуватися для зменшення навантаження і прискорення відгуку веб-додатків, ресурсоємні завдання можуть бути делеговані третій стороні через чергу.

3.7 Grafana

Grafana - це багатоплатформне рішення з відкритим кодом для аналітики та візуалізації даних, обчислення метрик та моніторингу програм через налаштовані панелі інструментів. У оновленні 2014 року був доданий інтерактивний інтерфейс для візуалізації різного роду графіків та сповіщення, коли служба підключена до підтримуваних джерел даних (рисунок 3.4).

Grafana може бути підключена до великої кількості різнорідних джерел даних, таких як Graphite або Influx DB.

Рисунок 3.4 — Графіки та метрики у Grafana

Як графічний інструмент, Grafana є одним з найкращих статистичних сервісів, який часто використовується в поєднанні з різними базами даних, які базуються на математичних конструкціях (Prometheus).

3.8 Висновки до розділу

При розробці системи збору даних та прогнозування було використано сучасні технології та інструменти для програмної реалізації модулів та розгортання їх інфраструктури.

Програмний код модуля збору статистики та модуля програмування написано мовами PHP та Python відповідно. Сховище даних – ElasticSearch. Взаємодія з іншими сервісами продукту здійснюється через сервіс шини повідомлень RabbitMQ. Для візуалізації даних обрано за стосунок Grafana. Усі компоненти запускаються у контейнерах віртуальної машини Docker.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Сервіс збору статистики та прогнозування реалізовано у вигляді двох модуля збору та збереження даних та модуля прогнозування. Їх взаємодію налагоджено через доступ до спільних даних у сховищі ElasticSearch.

4.1 Модуль збору статистики

Модуль збору статистики реалізовано на основі Nginx-PHP web-сервера, що функціонує у Linux середовищі Docker контейнера. Застосунок 'слухає' окрему, створену для нього чергу Event bus. Таким чином всі повідомлення сервісів, на які підписаний модуль, дублюються у черзі RabbitMQ – окрім основної, вони потрапляють у спеціальну чергу сервісу статистики (рисунок 4.1).

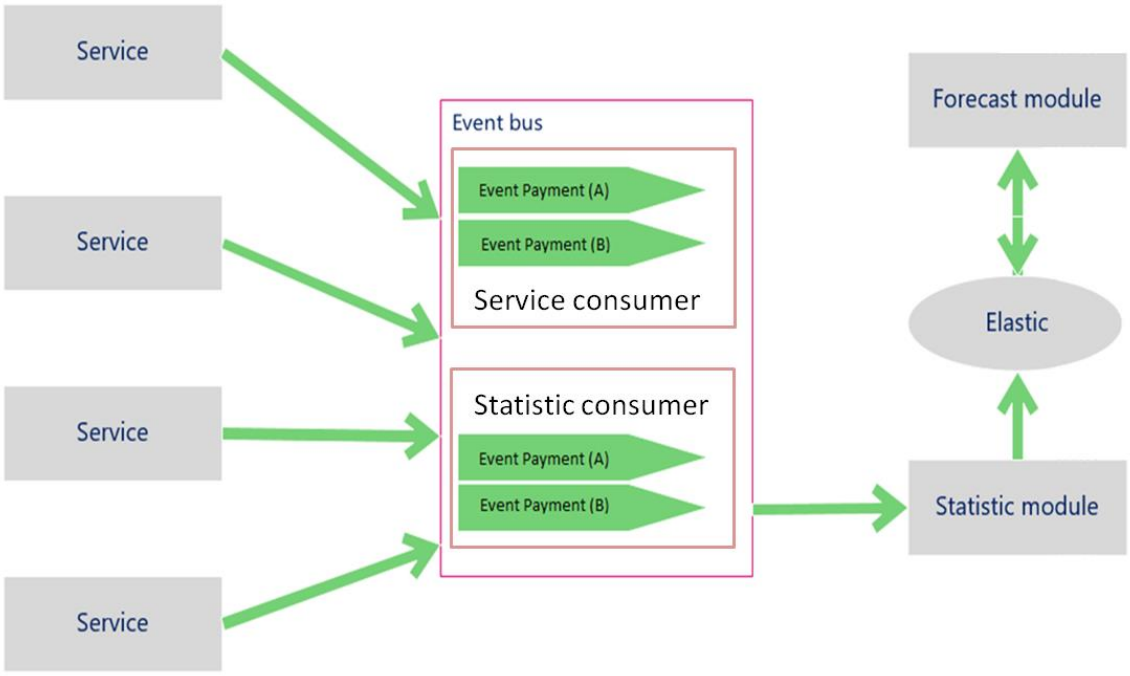

Рисунок 4.1 — Структура модуля

Сервер отримує з повідомлення про подію, якщо користувач увімкнув збір статистики для даного типу подій – дані про неї буде записано у базу даних ElasticSearch. Запис містить інформацію про код сервісу, що обробляв подію, тип події, статус (успішне чи ні завершення обробки події), час та мітку прогнозу (рисунок 4.2).

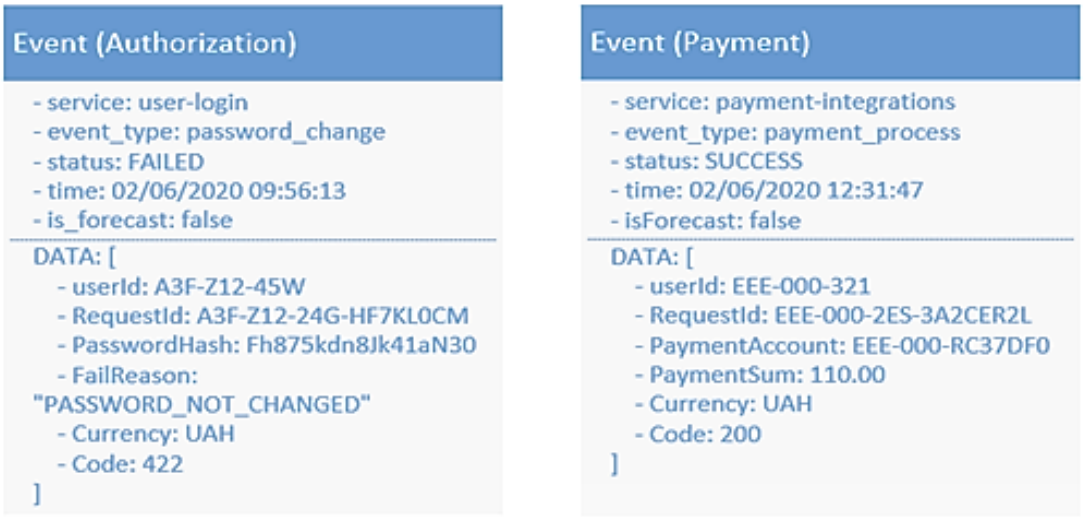

Рисунок 4.2 — структура записів про подію у ElasticSearch

Структуру програмного коду модуля збору даних виконано за шаблоном MVC (рисунок 4.3). Вхідна точка – контроллер. Він опрацьовує запити користувача на реєстрацію, перегляд, пошук та фільтрацію статистики. Всі сервіси, що використовують спільну шину повідомлень, фактично теж є користувачами модуля збору даних.

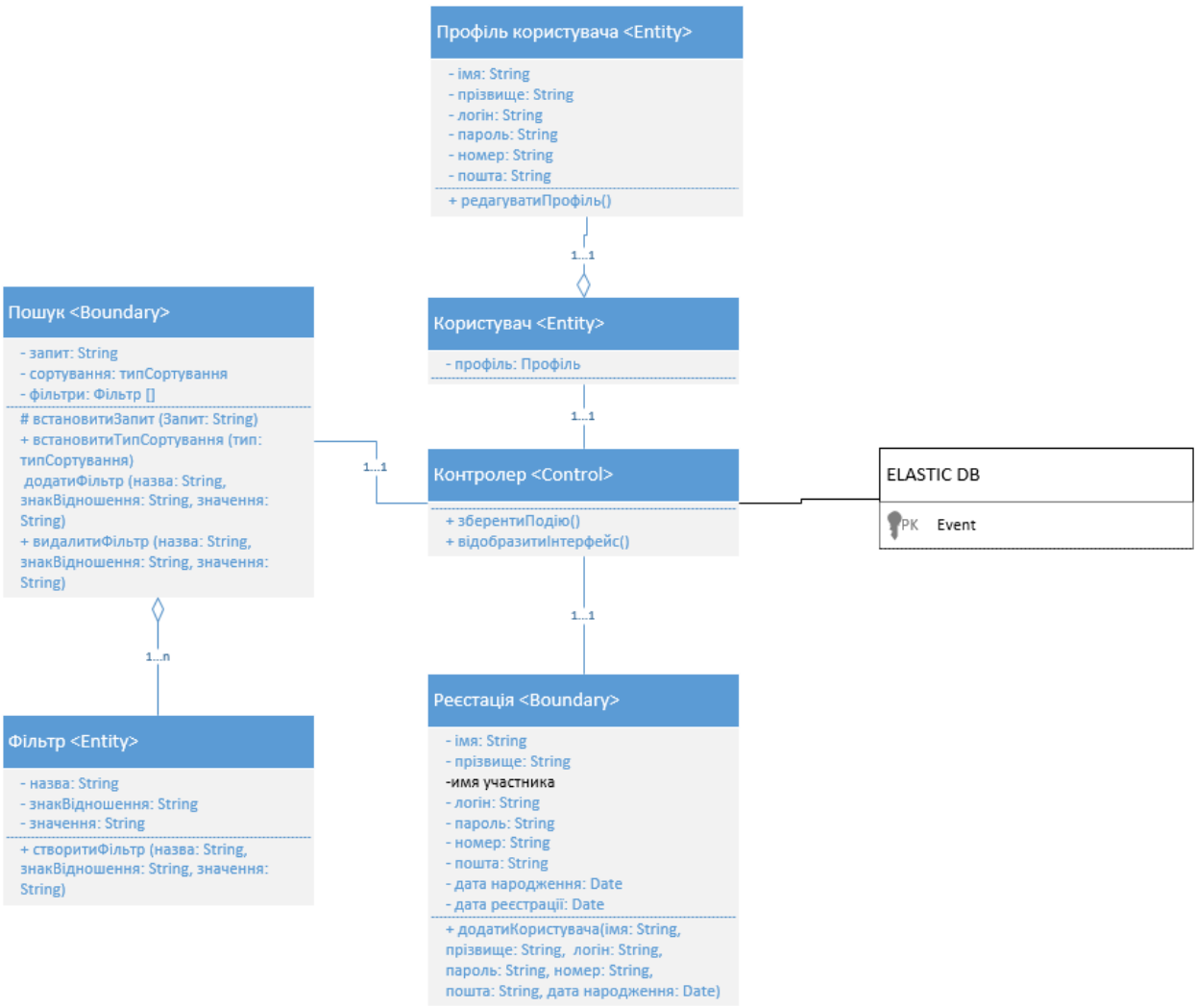

Рисунок 4.3 — UML діаграма класів

Щоб отримати доступ до даних статистики користувач повинен бути зареєстрований. Якщо при спробі увійти у сервіс в запиті відсутній токен авторизації – відбудеться перенаправлення(redirect) на сторінку реєстрації. Для авторизації потрібні логін користувача та пароль (рисунок 4.4).

Рисунок 4.4 — Вікно авторизації

Запити на візуалізацію опрацьовує застосунок Grafana. В ньому ж можна додати для відслідковування нові типи подій, фільтрацію.

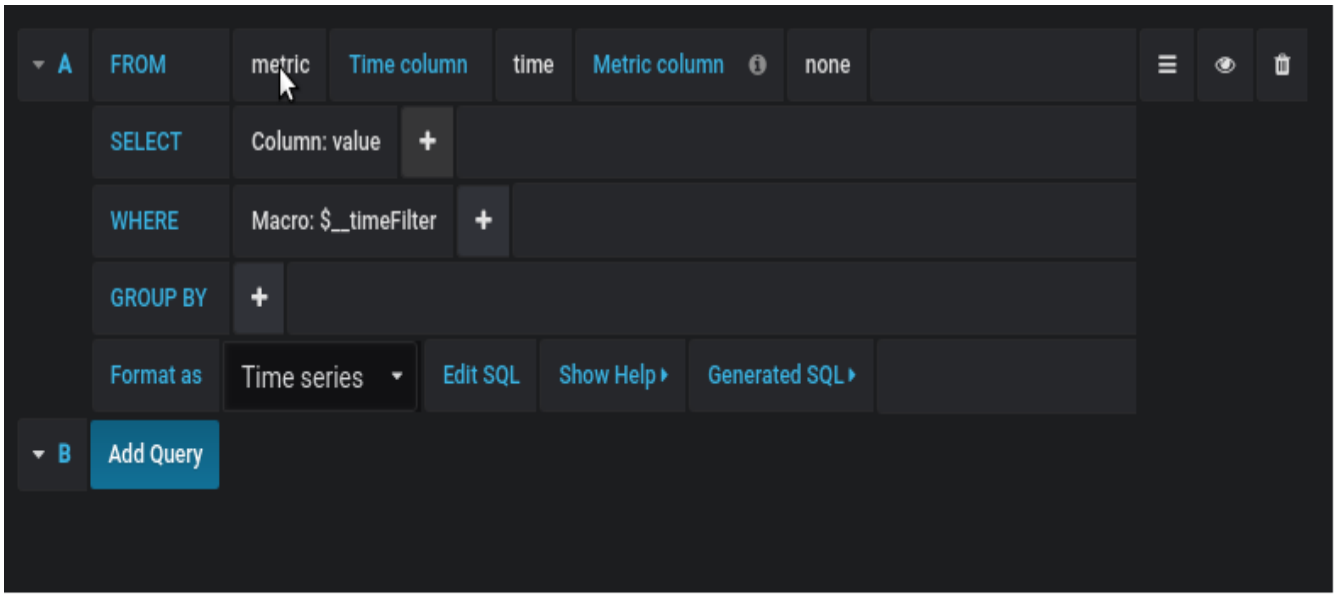

Рисунок 4.5 — Фільтри Grafana

Для відображення прогнозу користувачу потрібно обрати додаткові тип події, оригінальний графік та графік прогнозу (відфільтрований по параметру is_forecast) (рисунок 4.5). Графік прогнозу буде відмальовано далі по осі абсцис (рисунок 4.5). Одиниця вимірювання по осі ординант – кількість подій потрібного типу на сервісі.

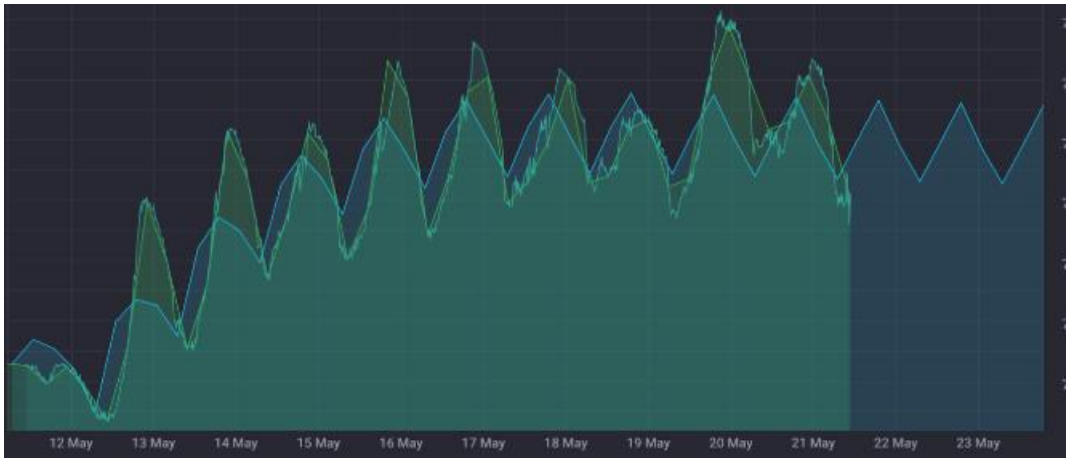

Рисунок 4.6 — приклад графіку прогнозу

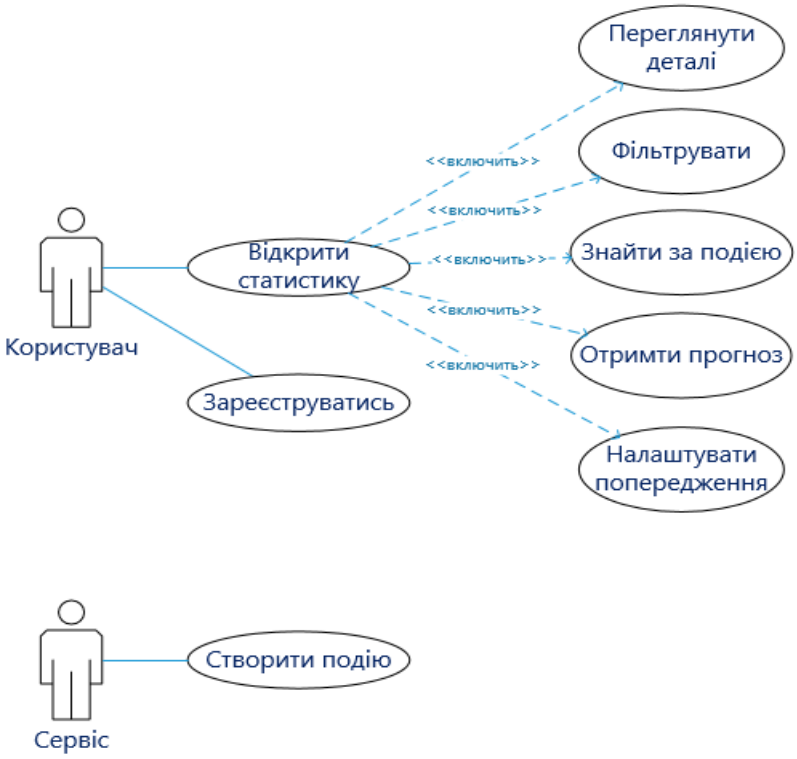

Рисунок 4.7 — UML діаграма прецедентів

Напряму з модулем збору статистики взаємодіє лише користувач через інтерфейс Grafana (рисунок 4.7). Grafana відображає дані, збережені в базі Elastic, але надає широкі можливості для налаштування форматів відображення, створення власних панелей моніторингу, комбінації датчиків, графіків та фільтрів даних.

4.2 Модуль прогнозування

Модуль прогнозування реалізовно у вигляді сервісу на мові Python, що включає у себе статистичну модель ARIMA та компоненти взаємодії з базою даних ElasticSearch. Запуск модуля відбувається ітеративно кожного дня. Система зчитує дані про подію, для якої генерується прогноз за останній доступний проміжок часу та передає їх до моделі. Результатом обробки даних моделлю є часовий ряд, з автозгенерованими прогнозними подіями (рисунок 4.8).

```
{
 "service": "user_login",
 "event_type": "password_change",
 "events" : [
    {"timestamp":"1590425370", "status":"SUCCESS"},
    {"timestamp":"1590425439", "status":"SUCCESS"},
    {"timestamp":"1590425501", "status":"SUCCESS"},
    {"timestamp":"1590425510", "status":"SUCCESS"},
    {"timestamp":"1590425521", "status":"SUCCESS"},
    {"timestamp":"1590425523", "status":"SUCCESS"},
    {"timestamp":"1590425535", "status":"SUCCESS"},
    {"timestamp":"1590425544", "status":"FAILED"},
    {"timestamp":"1590425549", "status":"SUCCESS"}
    ...
  ]
}

```

Рисунок 4.8 — результат роботи модуля прогнозування

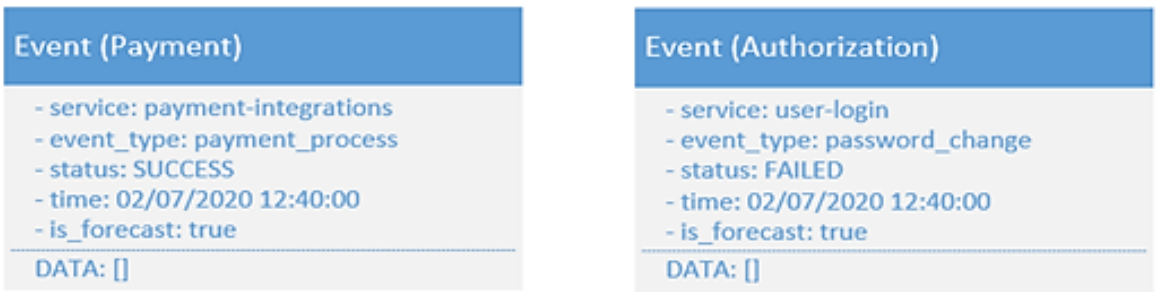

Рисунок 4.9 — структура даних згенерованої події у базі даних

Згненеровані події не містять додаткової інформації про джерело запиту, їх дата та час більші за поточні. Окрім цього, вони містять мітку маркер прогнозу (значення true у полі is_forecast). Модуль зберігає ці дані до бази даних ElasticSearch як і звичайні події, ініційовані сервісами продукту (рисунок 4.9).

4.3 Висновки до розділу

У результаті виконання дипломної роботи було створено модульну систему збору та обробки статистичних даних у складній розподіленій системі. Компонент обробки статистики отримує інформацію про події, які обробляються сервісами продукту, через шину повідомлень. Форматовані метрики зберігаються у базу даних ElasticSearch. Модуль прогнозування напряму не пов’язаний ні з модулем збору статистики, ні з іншими сервісами продукту. Він взаємодіє лише опосередково, через генерацію та збереження подій до сховища даних. Для візуалізації даних зі сховища використано застосунок Grafana. У ньому користувач може відобразити відображення метрик та прогнозів для конкретних сервісів та типів подій , налаштувати попередження при суттєвому відхиленні реальних даних від прогнозних оцінок.

5 РОБОТА КОРИТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для повноцінної роботи застосунку на веб сервері повинно бути встановлено віртуальну машину Docker. Так як всі контейнери з підмодулями працюватимуть всередині неї, вимоги за стосунку до апаратного забезпечення відповідають мінімальним вимогам віртуальної машини Docker, а саме:

- наявність 2.00 GB RAM
- 64 bit процесор
- Наявність 3.00 GB вільного дискового простору
- статична IP-адреса

Вимоги до програмного забезпечення:

- версія Linux kernel 3.10 або вище
- версія CS Docker Engine 1.10 або вище

5.1 Конфігурація системи

Система повинна отримувати інформацію про події зі спільної для продукту шини повідомлень. Для цього потрібно додати відповідне підключення до файлів конфігурації модуля збору статистики та вказати необхідну для підключення до черги інформацію (рисунок 5.1):

- driver – бібліотека, що використовується для здійснення підключення
- worker – тип скрипта-обробника події
- factory_class – файл скрипта, що генеруватиме об’єкти підключення
- host, port – URL чи IP адреса сервісу RabbitMQ та порт для підключення

- vhost – шлях до сервісу черг
- login, password – ідентифікація сервісу для доступу до черг
- queue – черга, на яку підписується сервіс. Для модуля збору статистики у застосунку RabbitMQ створено окрему чергу ‘statistic_server’
- sleep_on_error – час очікування при помилці черги у секундах
- ssl_params – параметри сертифікату SSL.

```
[
  'connections' => [
        'rabbitmq' => [

        'driver' => 'rabbitmq',

        'worker' => env('RABBITMQ_WORKER', 'default'),

        'factory_class' => Enqueue\AmqpLib\AmqpConnectionFactory::class,

        'host' => env('RABBITMQ_HOST', 'https://rabbitmq.product.com'),
        'port' => env('RABBITMQ_PORT', 5672),

        'vhost' => env('RABBITMQ_VHOST', '/rabbit'),
        'login' => env('RABBITMQ_LOGIN', 'statistic_server'),
        'password' => env('RABBITMQ_PASSWORD', 'statistic_server'),

        'queue' => env('RABBITMQ_QUEUE', 'statistic_queue'),

        'sleep_on_error' => env('RABBITMQ_ERROR_SLEEP', 5),

        'ssl_params' => [
            'ssl_on' => env('RABBITMQ_SSL', false),
            'cafile' => env('RABBITMQ_SSL_CAFILE', null),
            'local_cert' => env('RABBITMQ_SSL_LOCALCERT', null),
            'local_key' => env('RABBITMQ_SSL_LOCALKEY', null),
            'verify_peer' => env('RABBITMQ_SSL_VERIFY_PEER', true),
            'passphrase' => env('RABBITMQ_SSL_PASSPHRASE', null),
        ],

    ]
```

Рисунок 5.1 — конфігурація підключення до RabbitMQ

Створити окрему чергу повідомлень у сервісі RabbitMQ можна через графічний інтерфейс (рисунок 5.2). Для цього потрібно вказати ім’я нової черги та встановити параметри та обмеження (пріоритет, час очікування подій, максимальний розмір та дія при переповненні).

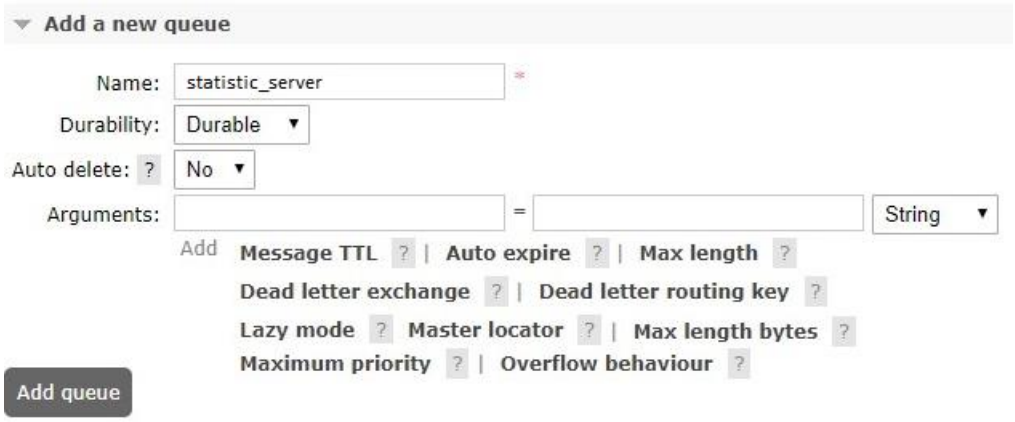

Рисунок 5.2 — інтерфейс додавання нової черги у RabbitMQ

Після цього нова черга сервісу статистики буде доступною у списку всіх черг (рисунок 5.3).

Overview			Messages			Message rates		
Name	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
statistic_server	D	running	0	0	0			
user_auth	D	running	151	32	1345			
export_files	D	running	45	1	397			
payments	D	running	27	0	142			

Рисунок 5.3 — список черг у RabbitMQ

Щоб події окрім основних черг дублювалися для модуля обробки сатистики, потрібно налаштувати типи подій, які потраплятимуть до черги (рисунок 5.4). Таким чином користувач контролює, яку інформацію отримає система збору сатистики.

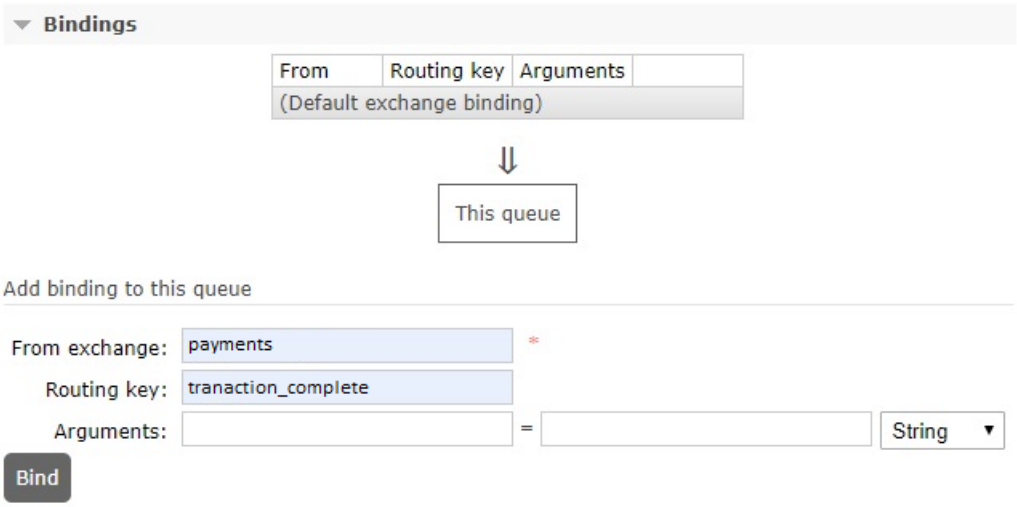

Рисунок 5.4 — прив’язка події до черги RabbitMQ

Користувач матиме можливість подальшого редагуання подій, на які підписана черга.

Рисунок 5.5 — налаштування подій черги

Присутня можливість додати для відслідковування нові події або вилучити існуючі (рисунок 5.5).

5.2 Взаємодія з системою

Користувач напряму не взаємодіє з модулем збору статитики та прогнозування. Ці компоненти є частиною інфраструктури продукту та працюють автоматично без втручання користувача. Під модуль збору даних в режимі реального часу зберігає інформацію про відслідковувані події до сховища. Модуль прогнозування відпрацьовує щоденно, генерує прогноз та записує його до сховища. Таким чином результатом роботи модуля є нові записи у базі даних.

Для зручності використання результатів роботи сервіс включає у себе Docker-контейнер з застосунком Grafana. Увійти до застосунку можна використовуючи логін з паролем або авторизуватися через сервіси Google, GitHub чи аккаунт Grafana (рисунок 5.6).

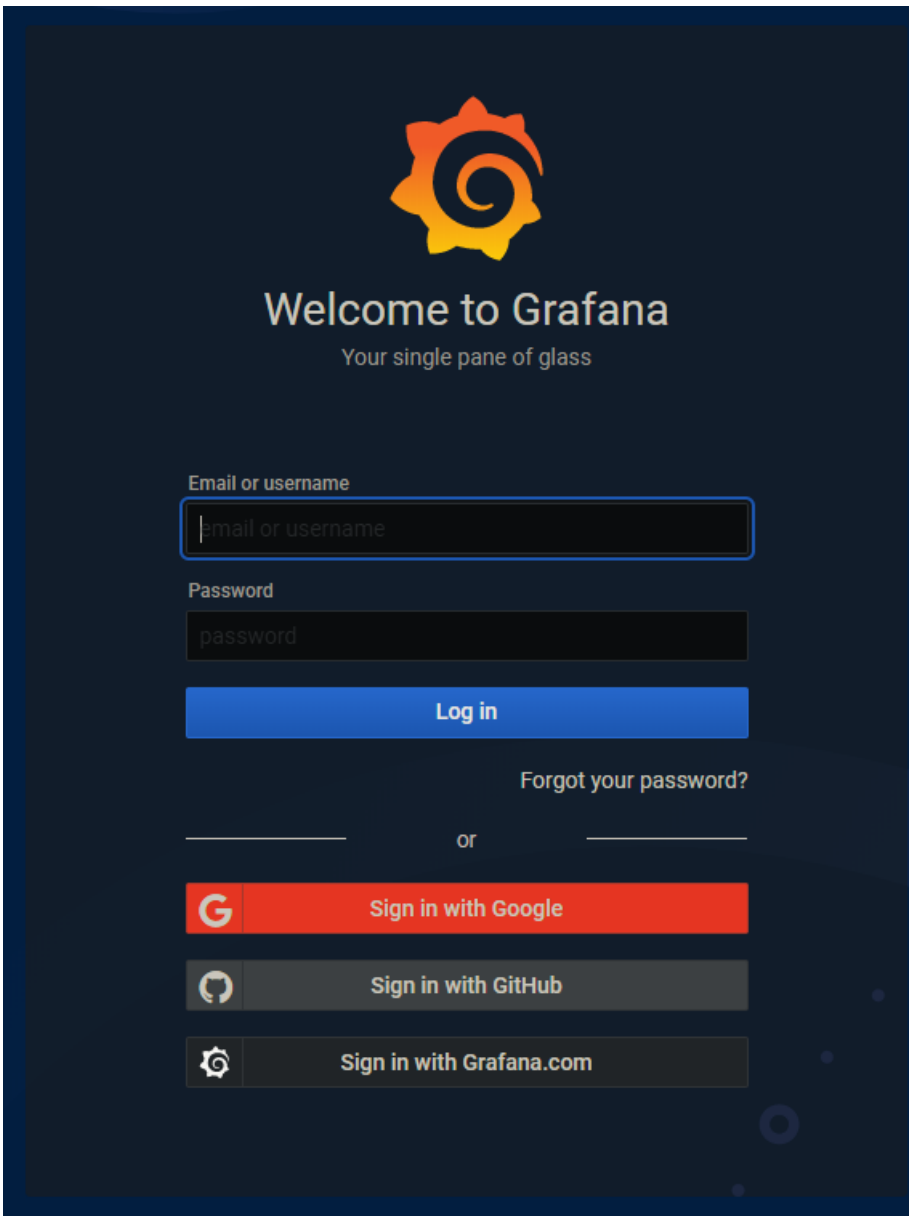

Рисунок 5.6 — вхід до за стосунку

Для відображення збережених даних користувачу потрібно для початку створити новий дашборд – полотно для відображення графіків та метрик (рисунок 5.7). Щоб співставити прогнозну частоту подій та реальну їх кількість у системі, потрібно додати фільтр даних. Вони реалізовані у вигляді SQL-подібного запиту. У

частині FROM вказується тип запису (event) та назву поля, у якому знаходиться час та дата події (time).

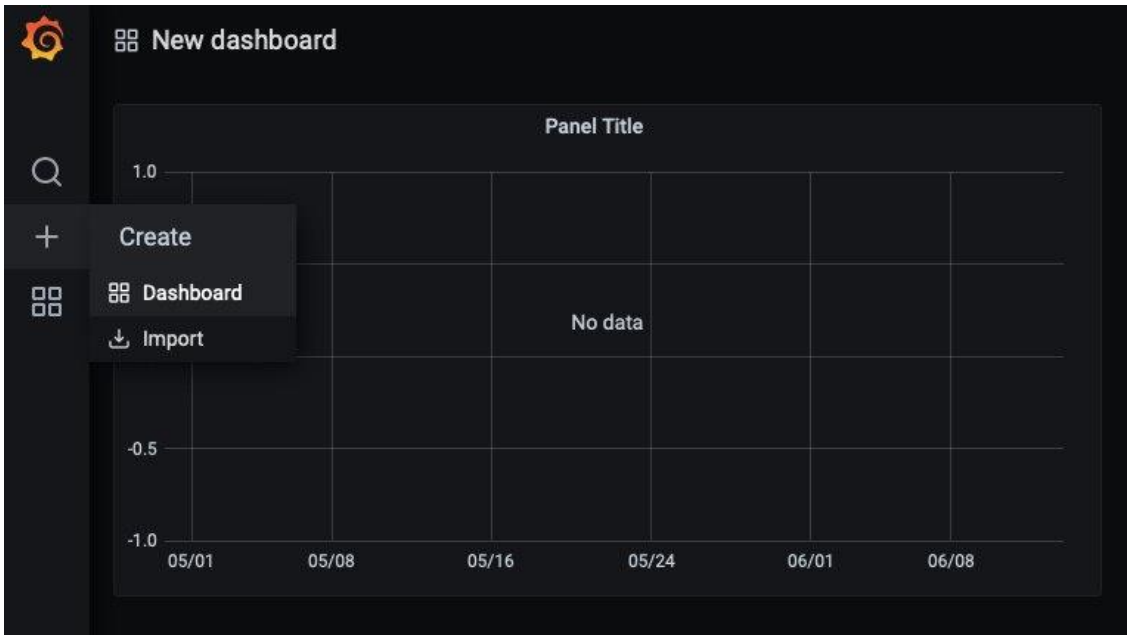

Рисунок 5.7 — додавання дашборду у Grafana

Структура є однаковою для всіх записів про події, створених модулем збору та прогнозування.

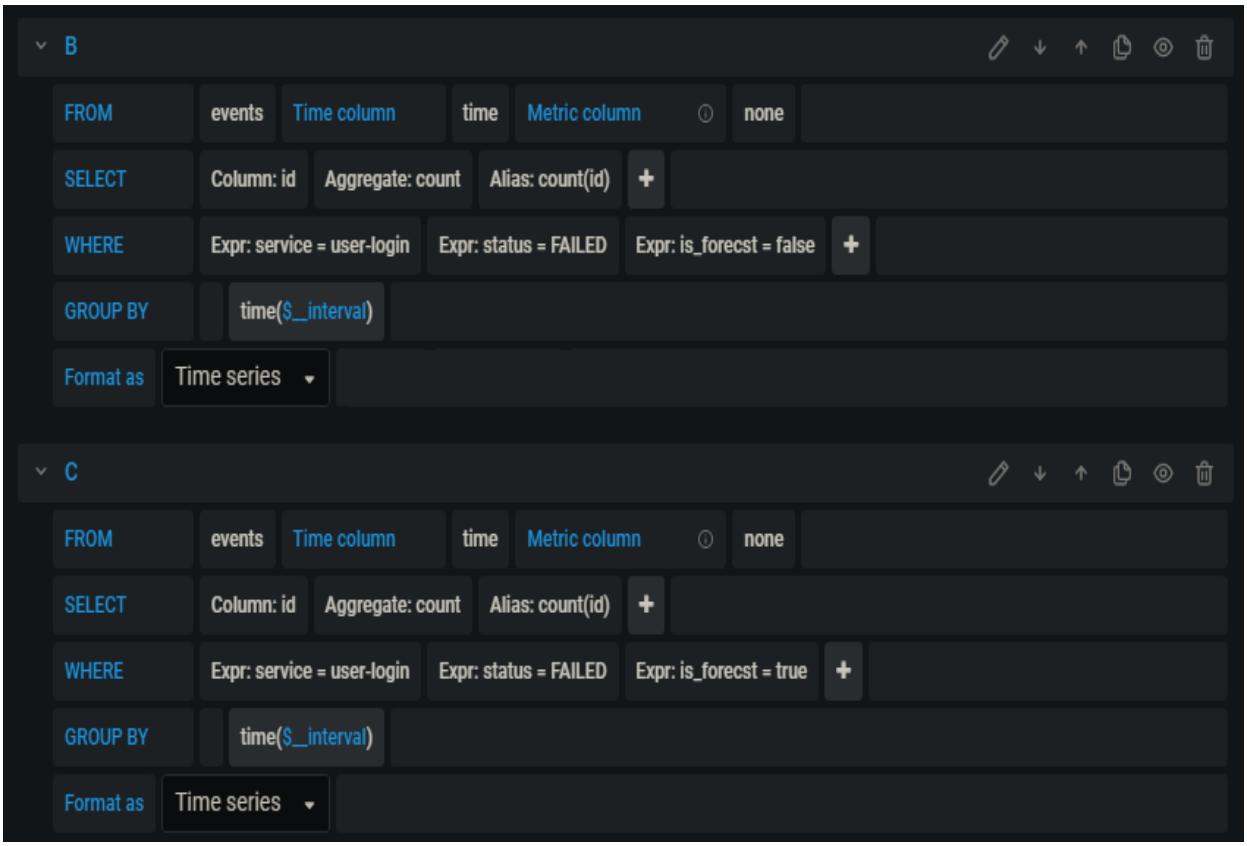

Рисунок 5.8 — фільтрація даних для відображення

У розділі SELECT вказуються поле ідентифікатора та агрегація за кількістю знайдених результатів. Компонент WHERE дає можливість вказати сервіс та тип події, для яких буде побудовано графік, а також обрати статус події, щоб відображати або успішні відпрацювання, або кількість помилок сервісу. В цій же частині фільтру вказується значення поля is_forecast. Щоб на дашборді одночасно були присутні графіки реальних даних і прогнозу потрібно додавати 2 фільтри, що відрізнятимуться лише значенням поля is_forecast, відповідно true для пронозу та false для справжніх подій (рисунок 5.8).

5.3 Висновки до розділу

Таким чином, для функціонування модуля, адміністратор шини повідомлень повинен одноразово створити для сервісу статистики окрему чергу RabbitMQ та налаштувати події, що потраплятимуть у цю чергу. Після цього дані про події та прогнозні оцінки будуть зберігатися до сховища даних ElasticSearch.

Застосунок Grafana надає можливість у будь-який момент візуалізувати статистику конкретного сервісу та типу події та спів ставити їх з прогнозною оцінкою. Також присутня можливість налаштувати попередження (alerting) при суттєвій відмінності реальних даних від прогнозної оцінки, що може свідчити про неполадки на сервісах продукту або невдале оновлення функціоналу, що зробило його менш привабливим для користувачів (наприклад зниження кількості події реєстрації користувача після зміни дизайну сторінки реєстрації). При появі нових типів подій або модифікацій структури існуючих користувачу достатньо відповідно додати новий дашборд або оновити фільтри на уже існуючого.

Висновки

Під час роботи над дипломною роботою було вдосконалено навички розробки самостійних модульних систем. Відпрацьовано уміння синхронізувати роботу компонентів складної розподіленої системи через шину повідомлень, набуто досвіду роботи з нереляційними (NoSQL) базами даних.

У результаті виконання дипломної роботи спроектовано та розроблено сервіс збору статистики та прогнозування подій у сервісах складної розподіленої системи. Спроектовані моделі було візуалізовано за допомогою UML діаграм.

Розроблена система дає можливість оцінювати якість роботи різних сервісів розподіленого продукту. На основі даних про очікувану та отриману кількість подій можна робити висновки про результативність змін та оновлень, застосованих у продукті. Наявність модуля прогнозування дозволяє ефективніше знаходити проблеми у роботі проекту.

Створений програмний продукт має переваги перед конкурентами, а саме відсутність потреби у модифікації самих сервісів та наявність модуля прогнозування. Окрім цього, модуль збору даних та статистична модель створено з урахуванням особливостей конкретного продукту, але сервіс можна інтегрувати і у інші розподілені системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. “Создание микросервисов”, Сэм Ньюмен. 2016.

2. “Статистические методы прогнозирования”, Дуброва, Т. А. 2003.

3. “Общая теория статистики”. И.И.Елисеева, М.М.Юзбашев. 2004.

4. “Анализ временных рядов и прогнозирование”, В.Н. Афанасьев, М.М. Юзбашев.

5. “Designing Event-Driven Systems”, Ben Stopford. 2018.

6. Docker: Orientation and setup — Режим доступу: https://docs.docker.com/

7. “Прикладная статистика. Основы эконометрики.”, Айвазян С.А. 2001.

ДОДАТОК 1

Моніторинг прогнозних оцінок ефективності роботи сервісів в складних розподілених системах

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТІ6162_20Б

Аркушів 2

Київ – 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6162_20Б 81-1	Записка Защик ТІ-61.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6162_20Б 12-1	event_processor.php	Обробник подій з шини повідомлень
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6162_20Б 12-2	emmit_events.php	Генератор подій
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6162_20Б 12-3	Queue.php	Файл конфігурації черги RabbitMQ
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6162_20Б 12-4	ForecastProcessor.py	Обробник прогнозування

ДОДАТОК 2

Моніторинг прогнозних оцінок ефективності роботи сервісів в складних розподілених системах

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТІ6162_20Б 12-1

Аркушів 10

Київ – 2020

```
// Конфігурація
 $params = config('connections.rabbitmq');

        // Підключення
$connection = new AMQPConnection();
$connection->connect();

        // Канал повідомлень
$channel = new AMQPChannel($connection);

        // Об'єкт отримання повідомлень
$exchange = new AMQPExchange($channel);
$exchange->setName('ex_hello');
$exchange->setType(AMQP_EX_TYPE_FANOUT);
$exchange->declare();

        // Підключення до черги
$queue = new AMQPQueue($channel);
$queue->setName('hello');
$queue->setFlags(AMQP_IFUNUSED | AMQP_AUTODELETE | AMQP_DURABLE);
$queue->declare();
$queue->bind($exchange->getName(), '');

        // Сервіс збереження подій
$eventStorage = new ElasticSearchEventStorage();
        // Форматувальник подій
$eventFormatter = new EventFormatter();

        // Обробка всіх повідомлень черги
while (true) {
    if ($envelope = $queue->get()) {
        $message = json_decode($envelope->getBody());
```

```
        // Форматування даних події
    $event = $eventFormatter->format($message)
        // зберігання даних події
    if (eventStorage->store($event)) {
        $queue->ack($envelope->getDeliveryTag());
    } else {
        // повторне надсилання у разі падіння
        $queue->nack($envelope->getDelivaryTag(), AMQP_REQUEUE);
    }
  }
}

        // Закриття підключення
$connection->disconnect();
```

```
[
        // Конфігурація
 $params = config('connections.rabbitmq');
        // Підключення
$connection = new AMQPConnection();
$connection->connect();
        // Канал повідомлень
$channel = new AMQPChannel($connection);

        // Об'єкт обміну подіями
$exchange = new AMQPExchange($channel);
$exchange->setName('statistic_server');
$exchange->setType(AMQP_EX_TYPE_FANOUT);
$exchange->setFlags(AMQP_IFUNUSED | AMQP_AUTODELETE);
$exchange->declare();

        // Підключення до черги
$queue = new AMQPQueue($channel);
$queue->setName('event_emmit'');
$queue->setFlags(AMQP_IFUNUSED | AMQP_AUTODELETE | AMQP_DURABLE);
$queue->declare();
$queue->bind($exchange->getName(), '');

        // Тіло події
$event = (new EventService())->getEvent($queue);

        // результат надсилання події
$result = $exchange->publish(json_encode(), $event');
        // перевірка успішності надсилання
if ($result)
   $logger->info('event_sent');
else
   $logger->error('event_failed);
```

```
        // Закриття підключення
$connection->disconnect();
```



```
[
  'connections' => [
      'rabbitmq' => [
         // Драйвер
      'driver' => 'rabbitmq',
         // Обробник
      'worker' => env('RABBITMQ_WORKER', 'default'),
         // Клас підключення
      'factory_class' => Enqueue\AmqpLib\AmqpConnectionFactory::class,
         // Адреса і порт сервісу черг
      'host' => env('RABBITMQ_HOST', 'https://rabbitmq.product.com'),
      'port' => env('RABBITMQ_PORT', 5672),

         // Дані авторизації
      'vhost' => env('RABBITMQ_VHOST', '/rabbit'),
      'login' => env('RABBITMQ_LOGIN', 'statistic_server'),
      'password' => env('RABBITMQ_PASSWORD', 'statistic_server'),
         // Назва черги
      'queue' => env('RABBITMQ_QUEUE', 'statistic_queue'),
         // Очікування при падінні
      'sleep_on_error' => env('RABBITMQ_ERROR_SLEEP', 5),
         // Параметри сертифікату SSL.
      'ssl_params' => [
         'ssl_on' => env('RABBITMQ_SSL', false),
         'cafile' => env('RABBITMQ_SSL_CAFILE', null),
         'local_cert' => env('RABBITMQ_SSL_LOCALCERT', null),
         'local_key' => env('RABBITMQ_SSL_LOCALKEY', null),
         'verify_peer' => env('RABBITMQ_SSL_VERIFY_PEER', true),
         'passphrase' => env('RABBITMQ_SSL_PASSPHRASE', null),
               ],]]
```

```
// Імпорт статистичних бібліотек, сервісу форматування події да їх збереження до хранилища.
import numpy as np
import pandas as pd
import event_service as events
import events_generator
import event_storage as storage
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA

// Отримання списку сервісів, для яких потрібно побудувати прогноз
services = events.get_services();

// Отримання типів подій для побудови прогнозу
events_list = events.get_events_by_services(services)

// підготовка даних для обробки моделлю
prepared_list = pd.read_csv(events.prepareCsv(events_list), parse_dates = ['Month'], index_col = ['Month'])
prepared_list.head()

// Сезонна декомпозиція часового ряду
decomposition = seasonal_decompose(prepared_list)

// Обробка даних моделлю
model = ARIMA(df_log, order=(2,1,2))
results = model.fit(disp=-1)

// Отримання прогнозу
predictions_ARIMA_diff = pd.Series(results.fittedvalues, copy=True)
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
```

```
predictions_ARIMA_log = pd.Series(df_log['Passengers'].iloc[0], index=df_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum, fill_value=0)

// Фільтрація лише прогнозних даних
predictions_ARIMA = np.exp(predictions_ARIMA_log)

// Генерація об’єктів для збереження до сховища
forecat_event_objects = generator.process_predictions_list(predictions)

// Збереження прогнозних подій
storage.save(forecast_event_objects)
```

ДОДАТОК З

Моніторинг прогнозних оцінок ефективності роботи сервісів в складних розподілених системах

Опис програмного модулю

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТІ6162_20Б 13-1

Аркушів 8

Київ – 2020

АНОТАЦІЯ

Розроблена система збору статистики та прогнозування складається з двох підсистем. Модуль збору даних зчитує інформацію про події на сервісі зі спільної черги RabbitMQ, форматує їх та зберігає до сховища даних ElasticSearch. Компонент прогнозування генерує список прогнозних подій на основі збережених даних за попередній проміжок часу.

Систему розроблено мовами програмування PHP та Python. Розгортання системи здійснюється в середовищі віртуальної машини Docker на базі операційної системи Linux.

Основними перевагами системи перед аналогами є використання спільної з іншими сервісами шини повідомлень та адаптованість моделі прогнозування під особливості конкретного продукту.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Розроблена система має назву “Модуль збору статистики та прогнозування подій у сервісах складних розподілених систем”.

Система представляє собою серверний додаток що розгортається на основі віртуальної машини Docker. Для використання модуля потрібно створити у спільній шині повідомлень продукту окрему чергу для модуля збору статистики та вказати шлях підключення до неї та дані авторизації у файлах конфігурації сервісу.

Модуль збору даних реалізовано мовою програмування PHP на основі фреймворка для веб розробки Laravel. Компонент прогнозування написано мовою програмування Python з використанням статистичної моделі ARIMA пакету statsmodels.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначенням системи є збір та збереження даних про події на сервісах складних розподілених систем з їх подальшою обробкою – створення прогнозів виникнення подій на цих сервісах на основі даних за попередній проміжок часу.

Для збору даних використано шину повідомлень RabbitMQ, що дозволяє не навантажувати сервіси додаванням логіки надсилання статистичних даних. Для прогнозування використано статистичну модель ARIMA, що дозволяє врахувати сезонність частоти виникнення подій та загальні тренди розвитку продукту.

Результатом роботи сервісу є збережені у сховищі ElasticSearch згенеровані модулем прогнозування події, які є продовженням часового ряду реальних подй на сервісах.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Розроблена система складається з двох незалежних частин – модуля збору даних та модуля прогнозування. Взаємодія між ними здійснюється через спільне сховище даних.

Модуль збору даних отримує повідомлення про подію, структурує її дані та записує їх до сховища Elastic.

Компонент прогнозування запускається щодня. Він зчитує дані зі сховища і генерує продовження часового ряду для кожного унікального сервісу, типу події та статусу події (Failed/Success). Згенеровані дані модуль зберігає у вигляді записів, аналогічних записам про реальні подій, але з міткою прогнозу та відсутністю додаткової інформації про подію.

ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для реалізації системи використано мови програмування PHP та Python. Середовище розробки – JetBrains PhpStorm. У якості інтерфейсу користувача використовується застосунок Grafana.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Вхідні дані:

1. сервіси та типи подій, що повинні відслідковуватися
2. вміст відслідковуваних подій за останній період

Вихідні дані:

1. записи подій на сервісах збережені у сховищі даних
2. згенеровані модулем прогнозування записи про події у мабутньому